Introducing TC53
ECMAScript Modules for Embedded Systems

Peter Hoddie
Chair, TC53
Co-founder, Moddable
Goal

- Bring benefits of software development on the web to embedded developers
  - Rapid development – modern programming language
  - Vendor neutral – code runs on many hardware platforms
  - Secure – safe operation, user privacy respected
  - Scalable – large, complex projects are more manageable
  - Open – APIs defined by collaboration of experts, not one single company
“ECMAScript Modules for Embedded Systems”

• JavaScript APIs
  • Common operations – I/O, networking, sensor, BLE, displays, security, energy management, etc.
  • Organized into software modules by operation
• Targeting wide range of embedded systems
  • Includes low cost embedded devices
• No changes to the JavaScript language for embedded
  • Exact same language as on the web
Standards are About Interoperability

- Many common capabilities in all microcontrollers
  - No common APIs
- Most silicon vendor has their own API
  - Expensive for silicon vendor to maintain
  - Difficult for software developers to learn
  - Rewrite software to move change silicon platform
- Standard APIs
  - Stop re-inventing APIs for common tasks
  - Increase code re-use
Design Priorities

- Efficient
  - Low CPU use, limited RAM use, small code size
- Secure
  - Protect users from privacy and safety vulnerabilities
- Portable
  - APIs work the same way across silicon vendors
- Approachable
  - Simple, consistent APIs
Licensing

- **Royalty free** working group
  - Contributors agree when joining TC53
- Anyone can implement the standard
  - No royalty payment
  - No licensing fee
- Similar policy as web technologies
  - JavaScript language (Ecma International)
  - HTML5 (W3C)
Contributors (partial)

- **Moddable** – efficient scripting and touch screen support
- **Monotype** – text handling, fonts, and text rendering
- **Whirlpool** – embedded system requirements, focus on safe operation
- **Michigan State University** – data precision for big data analysis
- **Agoric** – provably secure script execution
- **Bocoup** – sensors and robotics applications
- **LyTen** – portable drivers for sensors, displays, and energy harvesting
- **Bob Frankston** – open connectivity and open APIs
Roadmap – Overview

• **Input/Output** – drafting underway

• **Sensors** – committee discussions

• **Energy management** – contributor investigations

• **Secure ECMAScript** – contributor investigations

• **Display Drivers** – agreed future work

• **Network Protocols** – agreed future work
Roadmap

Input/Output (I/O)

• Common microcontroller I/O capabilities
  • Digital, Analog, I2C, SPI, UART/Serial, PWM, Network sockets,
  • “IO Class Pattern” provides common API for all I/O
  • API design guidance for other I/O types
  • “IO Provider Class Pattern” to access external I/O
    • GPIO expander, analog expander, network sensor, BLE sensor
Roadmap Sensors

- “Sensor Class Pattern” provides common API for all sensors
  - Access to unique features of each sensor
- Builds on I/O
- API design guidance for other sensor types
Roadmap
Energy Management

• Battery operation

• Efficient use of AC power (EU regulations)

• Capabilities
  • Deep sleep
  • Energy efficient execution modes
  • Power down unused internal subsystems & external components
Roadmap

Secure ECMAScript

- Sandbox for JavaScript code
- Necessary for large, complex systems
  - Code from many engineers, departments, companies, open source contributors
- Compartments restrict access to resources
  - Fully customizable security policy
  - Built on proven Object Capabilities model (OCAP)
  - Extremely efficient
- Working to standardize with Ecma TC39 (JavaScript language committee)
Roadmap

Future Work

• Display drivers
  • Reduce barriers to adding displays
  • Build on Input/Output

• Network protocols
  • HTTP, MQTT, WebSocket, mDNS, CoAP, etc.
  • TLS/SSL for secure communication
  • Build on network socket
TC53 is Unique

- Standardizing APIs
  - Most IoT standards focus on data formats and communication protocols
- Preparing for the future, not predicting it
  - With the right APIs, you can implement any data format or communication protocol
- Focus on JavaScript
  - Best language to build IoT products
  - Efficient development, secure and reliable, code, proven for communication
  - Unique security properties
Why JavaScript?
Most Popular Language

• Most widely used programming language today
  • Web pages
  • Web servers
  • Mobile apps
  • Desktop apps
• Natural to extend to embedded systems
Why JavaScript?

JSON

- Standard data interchange format
- Subset of JavaScript
  - Native data format of JavaScript
  - Easy and efficient to use in JavaScript code
- Common in IoT communication
Why JavaScript?
Unmatched Ecosystem

- Learning resources
- Skilled developers
- Development tools
Why JavaScript?
A Real Standard

• JavaScript is a formal international standard
  • Independent of any one company
  • Dozens of companies contribute to its evolution
  • Most scripting languages are not true standards (e.g. Lua, Perl, Python, Ruby)

• Multiple implementations
  • 6 full modern JavaScript engines
  • Many specialized engines
Why JavaScript?
Stable. By Design.

• “Don’t break the web”

• 24 years of backwards compatibility

• Language stability critical for embedded products
  • Embedded products (washing machine, thermostat, LED light bulb) have 10+ year life span
  • Software updates may be necessary during that time
  • Backwards incompatible language changes make updates more difficult
Why JavaScript?

Efficient on Embedded

- JavaScript was relatively slow
  - V8 engine from Google changed that
  - New techniques boosted speed by orders of magnitude
  - Created new possibilities, reshaped the modern web
- JavaScript was too resource intensive for embedded
  - XS engine from Moddable changes that
  - New techniques reduce resource use by orders of magnitude
  - Creates new possibilities, revolutionizing embedded products (we hope!)
Patrick Soquet
Impacts
Impacts

- Linux created a de-facto software standard for certain kinds of computing

- TC53 aims to create a software Standard for embedded systems.
Benefits

- Software standards benefit the entire ecosystem

- **Microcontroller makers** – create less proprietary software, leverage design work of standard

- **Peripheral makers** (sensors, actuators, displays) – focus on great product not porting drivers

- **Software developers** – build on top of a solid, well designed platform. Higher quality result in less time.

- **Users** – more reliable, secure, and innovative products
One More User Benefit
One More User Benefit Apps
Apps

- User installable apps on embedded systems
  - Change behavior and features of products
- JavaScript makes it possible
  - Just like adding scripts to web pages
  - Independent of silicon architecture
  - Safe and secure with Secure ECMAScript
- Opens up new world of possibilities
  - Product manufacturers can leverage third party developers
Get Involved!
Conclusion

• Software for embedded systems is ready to be standardized
  • Huge benefits for entire ecosystem
  • Ecma TC53 is bringing the success of JavaScript on the web to embedded systems
    • Proven technology and processes
• Now is the time to begin
Thank you!